MULTIMEDIA UNIVERSITY®

STUDENT ID NO

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 2, 2016/2017

THP7021 – HIGH PERFORMANCE COMPUTING AND BIG DATA

(All sections / Groups)

10th FEBRUARY 2017
8.00 p.m - 10.00 p.m
(2 Hours)

INSTRUCTIONS TO STUDENT

1. This Question paper consists of 5 pages with 4 Questions only.
2. Attempt **ALL FOUR (4)** questions. All questions carry equal marks and the distribution of the marks for each question is given.
3. Please print all your answers in the Answer Booklet provided.

Question 1 [10 marks]

Given this serial programme:

```
double f(double x); // A very computation-intensive
fuction.
// Assume that START, END and the function f are defined
// somewhere.

int main(int argc, char *argv[])
{ double total = 0, x;
  int partitions;
  double slice;

  printf("How many partitions? "); fflush(stdout);
  scanf("%d", &partitions);
  slice = (END-START)/partitions;
  for (x = START + (slice/2); x < END; x = x + slice)
    total = total + f(x);
  total = total * slice;

  printf("The integration is %1.20f\n", total);
}
```

We have covered several ways to do parallel processing – MPI, OpenMP, Pthreads, and Hadoop.

a) What kind of situation would it be better to convert this program into an MPI program, as opposed the other ways we covered? Explain why this is the best choice in the situation you described, and why each of the other three are not suitable. [5 marks]

b) What kind of situation would it be better to convert this program into an openMP program, as opposed the other ways we covered? Explain why this is the best choice in the situation you described and why each of the other three are not suitable. [5 marks]

Continued...

Question 2 [10 marks]

Convert the serial program from question 1 into an OpenMP programme. [10 marks]

OpenMP quick reference

Constructs

#pragma omp parallel for [shared(vars), private(vars), firstprivate(vars), lastprivate(vars), default(shared|none), reduction(op:vars), copyin(vars), if(expr), ordered, schedule(type[,chunkSize])]

#pragma omp parallel sections [shared(vars), private(vars), firstprivate(vars), lastprivate(vars), default(shared|none), reduction(op:vars), copyin(vars), if(expr)]

#pragma omp parallel [shared(vars), private(vars), firstprivate(vars), lastprivate(vars), default(private|shared|none), reduction(op:vars), copyin(vars), if(expr)

Directives

shared(vars)
private(vars)
firstprivate(vars)
lastprivate(vars)
default(private|shared|none)
reduction(op:vars)
copyin(vars)
if(expr)
schedule(type,[,chunkSize])
nowait

Synchronization/Locking Constructs

#pragma omp master
#pragma omp critical
#pragma omp barrier
#pragma omp atomic
#pragma omp flush[(vars)]

Settings and Control

int **omp_get_num_threads**()
int **omp_get_thread_num**()
int **omp_in_parallel**()
int **omp_get_max_threads**()
int **omp_get_num_procs**()
int **omp_get_dynamic**()
int **omp_get_nested**()
double **omp_get_wtime**()
double **omp_get_wtick**()
void **omp_set_dynamic**(int)
void **omp_set_nested**(int)

Environment Variables

OMP_NUM_THREADS
OMP_SCHEDULE

Continued...

Question 3 [10 marks]

Convert the serial program from question 1 into an MPI programme. [10 marks]

MPI Quick Reference

Environmental Management:
int **MPI_Init**(int *argc, char **argv[])
int **MPI_Finalize**(void)
int **MPI_Initialized**(int *flag)
int **MPI_Finalized**(int *flag)
int **MPI_Comm_size**(MPI_Comm comm, int *size)
int **MPI_Comm_rank**(MPI_Comm comm, int *rank)
int **MPI_Abort**(MPI_Comm comm, int errorcode)
double **MPI_Wtime**(void)
double **MPI_Wtick**(void)

Blocking Point-to-Point-Communication:
int **MPI_Send** (void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
int **MPI_Recv** (void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
int **MPI_Probe** (int source, int tag, MPI_Commcomm, MPI_Status *status)
int **MPI_Get_count** (MPI_Status *status,MPI_Datatype datatype, int *count)
int **MPI_Sendrecv**(void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)
int **MPI_Sendrecv_replace**(void *buf, int count, MPI_Datatype datatype, int dest, int sendtag, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

Collective Communication:
int **MPI_Barrier** (MPI_Comm comm)
int **MPI_Bcast** (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
int **MPI_Gather** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
int **MPI_Gatherv** (void *sendbuf, intsendcount, MPI_Datatype sendtype, void*recvbuf, int recvcount_array[], int displ_array[], MPI_Datatype recvtype, int root, MPI_Comm comm)
int **MPI_Scatter** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
int **MPI_Scatterv** (void *sendbuf, int sendcount_array[], int displ_array[] MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
int **MPI_Allgather** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
int **MPI_Allgatherv** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount_array[], int displ_array[], MPI_Datatype recvtype, MPI_Comm comm)
int **MPI_Reduce** (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
int **MPI_Allreduce** (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
int **MPI_Reduce_scatter** (void *sendbuf, void *recvbuf, int recvcount_array[], MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
int **MPI_Op_create** (MPI_User_function *func,int commute, MPI_Op *op)
int **MPI_Op_free** (MPI_Op *op)

Wildcards:
MPI_ANY_TAG, MPI_ANY_SOURCE

Basic Datatypes:
MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE, MPI_PACKED

Predefined Groups and Communicators:
MPI_GROUP_EMPTY, MPI_GROUP_NULL, MPI_COMM_WORLD, MPI_COMM_SELF, MPI_COMM_NULL

Reduction Operations:
MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND, MPI_LOR, MPI_LXOR

Continued...

Question 4 [10 marks]

a) In Hadoop, what is map? [2 marks]

b) In Hadoop, what is reduce? [2 marks]

c) You have a server farm that has millions of web pages. You need to figure out which are the top ten most popular web pages. How would you do this with Hadoop? (The answer is only expected to be a few sentences – you do not have to write an essay.) [6 marks]

End of Paper